# 3D Visualization of Color Image Histograms

**Paula J. Reitan**[1]

*Department of Computer Science, University of Maryland Baltimore County, Baltimore, Maryland 21250, United States*

**Abstract**

Color quantization is a useful method for obtaining high-quality compressed images for storage and subsequent distribution via the World Wide Web. The first step of most color quantization techniques is to create a histogram of the colors in the truecolor image. The dramatic differences in time and space efficiency of various data structures used to represent truecolor image histograms are visually explored in 3D. Spatial subdivisions use buckets to store colors in a portion of the RGB cube. Linked lists are the typical data structure used for buckets; we propose using red-black trees and treaps. Our visualizations highlight the clustering property of spatial subdivision methods.

*Keywords:* 3D tree drawing, 3D Visualization, 3D data structures, color image histograms, color image quantization.

## 1. Introduction

In color quantization a truecolor image is irreversibly transformed into a color-mapped image consisting of $K$ carefully selected representative colors. When Heckbert proposed the color quantization problem in his seminal SIGGRAPH paper [6], most graphic workstations had CRT monitors with an 8-bit frame buffer, capable of displaying only $K = 2^8 = 256$ colors at a time. With the advent of the Internet and multimedia eras, 24-bit frame buffers have become more commonplace; however, the same has made the need for efficient ways to represent images more important. Color quantization is a useful method for obtaining high-quality compressed images for storage and subsequent distribution via the World Wide Web. For a survey of some of the more popular color quantization techniques see [15, 20, 19 and 12].

The first step of most color quantization techniques is to create a histogram of the colors in the truecolor image. Histogramming truecolor images is a basic component of many statistical image processing techniques including color quantization. The purpose of our color quantization research is to discover better heuristics for constructing color quantizers by visualizing the histograms, animating the quantization process, and visually comparing the resulting quantizers. In the process of building the software to visualize truecolor image histograms, it was found that portions of the software also serve nicely as an educational tool for illustrating the

[1] Email: reitan@scs.usna.navy.mil

space and time requirements of various 3D data structures. The goal of this paper is to present the pedagogical value of our aesthetically appealing visualizations of truecolor image histograms.

## 2. Related Work

There are numerous sources of work related to visualization of data structures in the field of graph drawing. An excellent annotated bibliography for theoretical and applied graph drawing papers was written by Di Battista, Eades, Tamassia, and Tollis [3]. In addition, Ding and Mateti surveyed [4] the data structure literature and provided a nice set of aesthetic criteria for drawing data structure diagrams in 2D. They proposed an algorithmic framework for drawing data structures using a weighted preference system to resolve conflicts between various aesthetic criteria. Although their drawings were strictly 2D, many of their aesthetic criteria are valid or extend nicely to 3D. Reingold and Tilford also proposed an algorithm for aesthetically drawing binary trees in 2D [14].

Over the past several years, more and more researchers have been exploring advances in 3D graphics hardware and software to produce graph drawings in 3D. Cone trees, a method for drawing hierarchical graphs in 3D, were first introduced by Robertson, Mackinlay and Card [16] as part of their Information Visualizer. The cones are formed by making each internal node of the tree the apex and placing their children around its base. All cones at the same level have the same diameter base, and the diameter of cone bases reduces with depth. The body of each cone is shaded transparently to aid the user in identifying subtrees. Garg and Tamassia describe GIOTTO3D, a system for visualizing hierarchical structures in 3D [5]. Whereas in 2D graph drawing, the hierarchical structure of a graph is typically indicated by the *y*-coordinate of the vertices, Garg and Tamassia propose using the *z*-coordinate, thus providing more flexibility in placing vertices in the *xy*-plane. Najork and Brown demonstrate the power of their 3D animation system, Obliq-3D, by drawing a *k*-d tree, for *k*=2, in 3D [10]. In this paper we provide a 3D visualization of a *k*-d tree, for *k*=3. Since presenting our visualizations at COMPUGRAPHICS ’97 [13], the author has become aware of unpublished work by Bokhari, Crockett and Nichol [2] which provides a visualization of truecolor histograms much like our cloud representation.

## 3. Truecolor Image Histograms

A digital image is digitized spatially so that it can be displayed on a raster display using *W* columns and *H* rows of pixels. A color digital image is digitized in color as well so that it can be displayed on a CRT monitor using the hardware-oriented RGB color space. The truecolor images discussed in this dissertation are stored in Targa's Truecolor image format [9]. The Targa format represents each image pixel using 8-bits of precision in each of the red, green, and blue color directions:

$$\mathrm{RGB} = \{ (r, g, b) \mid r, g, b \in \mathbb{Z}_{256} \}. \tag{1}$$

Thus RGB denotes the set of $256^3$ (16M) possible colors in a truecolor image[2]. It is generally accepted that truecolor images require a minimum of 8-bits of precision in each of the red, green, and blue color directions. A truecolor image $f$ is defined as a total function:

$$f\colon \mathbb{Z}_W \times \mathbb{Z}_H \to C, \tag{2}$$

where $f(x, y)$ is the RGB color of the pixel at column $x$, row $y$ of image $f$ and $C = \{ \boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_N \} \subseteq \mathrm{RGB}$. Thus $C$ is the set of $N$ unique colors present in image $f$. The histogram $H_f$ of a truecolor image $f$ is now defined as a total function:

$$H_f : \mathrm{RGB} \to \mathbb{N} \tag{3}$$

where $H_f(\boldsymbol{c})$ is the number of pixels in the image $f$ with color $\boldsymbol{c}$. Note that $H_f$ does not contain any spatial information about the pixels in $f$. Clearly, $\sum_{c \in C} H_f(c) = W \cdot H$ .

| | $\boldsymbol{W \cdot H}$ | $\boldsymbol{N_8}$ | $\boldsymbol{N_7}$ | $\boldsymbol{N_6}$ | $\boldsymbol{N_5}$ |
|---|---|---|---|---|---|
| *Windsails* | 393,216 | 86,008 | 72,663 | 26,233 | 5,873 |
| Max | 393,216 | 216,200 | 132,703 | 41,802 | 8,442 |
| Min | 65,536 | 856 | 848 | 821 | 490 |
| Average | 329,045 | 49,934 | 34,673 | 11,525 | 2,652 |

Table 1. Truecolor image test set statistics.

As part of our color quantization research we established a test set of 27 truecolor images. In Tables 1-6 we provide statistics specifically for the truecolor image *Windsails* ([17], Color Plate A) and summary statistics for the entire 27 image test set. Column $N_8$ of Table 1 summarizes the number of unique colors typically found in truecolor images. By cutting the (8-$p$) least significant bits from each of the R, G, B components, the RGB color space is uniformly quantized to a smaller color space denoted as $\mathrm{RGB}_p$. The number of unique $\mathrm{RGB}_p$ colors in a truecolor image will be denoted as $N_p$. Table 1 shows how such bit-cutting reduces the number unique colors

[1] The RGB color space is chosen simply because it is the hardware-oriented color space used by most display devices. However, the RGB color space is not ideal because it is not perceptually uniform. That is, two pairs of equidistant RGB colors may not be perceived as being “equally different”.

in truecolor images, thus saving space at the expense of accuracy. Many color quantization researchers find $\text{RGB}_5$ to be a satisfactory tradeoff between space, time, and accuracy. However, Shufelt [18] showed that working in $\text{RGB}_{p<7}$ can noticeably impair the quality of the quantized images. Thus, part of our color quantization research is to discover methods for making it more practical to work in full 24-bit precision.

## 4. The Data Structures

In this section, we will now examine various data structures used to store truecolor histograms. All of the histograms presented in this paper are for the truecolor image *Windsails* displayed in Color Plate A. Our 3D visualizations show the dramatic impact the choice of data structure has on time and space efficiency.

As part of our color image quantization research, we are developing a C++ program called LindyHop. All of the histograms presented in this paper were produced using LindyHop. LindyHop allows the user to view truecolor images, store truecolor image histograms in one of several different data structures, and to interactively choose among numerous algorithms to render the histograms in 3D space using a perspective projection. Thus, we are able to present more data in less screen space. To aid the user in viewing the 3D histogram, LindyHop provides interactive rotation, translation, and scaling.

### *4.1 Performance Analysis*

Given the large number of colors found in typical truecolor images, we are in search of a space efficient 3D histogram data structure that also supports fast insertion of pixels, query for $H_f$, and rendering. Therefore, the following performance measures are of interest: [3]

- *Space utilization* ($\lambda$) measures how well the data structure uses allocated memory.
- *Insertion time* ($T_I/t_I$) measures how much time is required to insert a pixel into the histogram.
- *Build time* ($T_B/t_B$) measures how much time is required to build the histogram.
- *Query time* ($T_Q/t_Q$) measures how much time is required to search the histogram for all image pixels.
- *Render time* ($T_R/t_R$) measures how much time is required to render a 3D visualization of the histogram.

[2] Analytical times will be represented by $T$; empirical times by $t$. Empirical times given in this paper are in seconds and were gathered on a SUN Ultra 1 with 128 MB of memory running under a light user load.

- The *height* ($h$) of hierarchical structures provides an upper bound for insertion and query times.

Because the shape of the dynamic data structures discussed in this section depend on the distribution of the colors in the image and the order in which they are inserted, it is difficult to analytically determine their average case performance. Thus, we will resort to empirical timing and structural data to measure the performance of these data structures.

## *4.2 3D Array*

Perhaps the most intuitive data structure class for RGB data is a 3D array of $2^p$x$2^p$x$2^p$ natural numbers used to store $H_f$. Thus $T_I = \Theta(1)$ and $T_Q = \Theta(W{\cdot}H)$. Table 2 and Color Plates B and C highlight the two main drawbacks of 3D arrays: low space utilization and high render times. Table 2 shows that space utilization ($\lambda = N_p / |\mathrm{RGB}_p|$) becomes unacceptably low for $p > 5$. Since it requires $\Theta(|\mathrm{RGB}_p|)$ time to create and initialize a 3D array and $O(W{\cdot}H)$ time to insert the $W{\cdot}H$ pixels of $f$, $T_B = \Theta(|\mathrm{RGB}_p| + W{\cdot}H)$. Thus, as $p$ increases, the time to initialize the 3D array becomes a more significant term of $\boldsymbol{T_B}$.

Color Plate B illustrates the low space utilization of 3D arrays by visually showing that a great deal of the color cubes space is not used. Additionally, the depth cues provided by the bounding cube's interpolated color and perspective tapering width help the user to properly orient the 3D visualization. To gain more insight into the nature of the histogram, Color Plate C shows how the sizes of the histogram cell may be scaled by $\log_2(H_f)$. To render a 3D array, we must traverse the entire 3D array and render each non-empty cell. Thus, $\boldsymbol{T_R} = \Theta(|\mathrm{RGB}_p| + N_p)$. Experimentally, we found that the size of $|\mathrm{RGB}_p|$ prohibits interactive speeds for $p > 6$.

So we conclude that from a space and time perspective, 3D arrays are an excellent choice for $\mathrm{RGB}_{p\leq 5}$. However, when greater accuracy is required ($\mathrm{RGB}_{p>5}$), we need a dynamic data structure whose space utilization and render time scales better with respect to $p$.

| | **RGB$_8$** | | | **RGB$_7$** | | | **RGB$_6$** | | | **RGB$_5$** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\lambda$ | $t_B$ | $t_Q$ | $\lambda$ | $t_B$ | $t_Q$ | $\lambda$ | $t_B$ | $t_Q$ | $\lambda$ | $t_B$ | $t_Q$ |
| *Windsails* | 0.0051 | 9.050 | 4.510 | 0.0346 | 5.520 | 4.440 | 0.1001 | 4.880 | 4.380 | 0.1792 | 4.700 | 4.350 |
| Max | 0.0129 | 9.120 | 4.540 | 0.0633 | 5.590 | 4.450 | 0.1595 | 4.890 | 4.380 | 0.2576 | 4.700 | 4.360 |
| Min | 0.0001 | 4.730 | 0.740 | 0.0004 | 1.310 | 0.720 | 0.0031 | 0.850 | 0.720 | 0.0150 | 0.780 | 0.720 |
| Avg | 0.0030 | 8.132 | 3.748 | 0.0165 | 4.628 | 3.707 | 0.0440 | 4.056 | 3.664 | 0.0809 | 3.925 | 3.643 |

Table 2. 3D array statistics for the image test set.

## *4.3 Binary Search Trees*

In this section we will discuss three forms of binary search trees (BSTs) and will use them as bucket structures for the spatial subdivision techniques discussed later. The nodes of BST histograms are ordered pairs, ($\boldsymbol{c} \in \mathrm{RGB}_p$, $count \in \mathbb{N}$), where $count = H_f(\boldsymbol{c})$. BSTs require that a total order exist for its elements. The following function converts $\boldsymbol{c} \in \mathrm{RGB}_p$ into a unique unsigned integer:

$$\textbf{key}(\, \boldsymbol{c} \,) = (\, \boldsymbol{c}.b \ll 16) \mid (\, \boldsymbol{c}.g \ll 8) \mid \boldsymbol{c}.r. \tag{4}$$

The `key` function is used to order the nodes of BST histograms. Color Plates D-H show the histogram of sailboats stored in three different forms of BSTs: traditional, AVL, and treap. We draw an arbitrary binary tree in 3D by introducing a natural extension to the 2D drawing algorithm implemented in LEDA [**7**]. Our binary tree drawing algorithm implements the following aesthetics:

- Nodes on the same [odd/even] level are aligned in the [*x*/*z*] direction.
- On [odd/even] levels, left subtrees are positioned to the [left/behind] their parent; right subtrees are positioned to the [right/in front].
- Parents on [odd/even] levels are centered in the [*z*/*x*] direction between its two children.
- Odd level edges between [left/right] subtrees and their parents are [light blue/blue]; even level edges are [light green/green]. This use of color helps the user to orient the visualization in 3D space.
- Subtrees are drawn isomorphically and symmetrically.

Color Plate D highlights the drawback of using a traditional BST: the height of the BST is unnecessarily large because the BST is very unbalanced. Red-black and AVL trees are alternative approachs to building BSTs such that the resulting BST is height-balanced. Color Plates E and F illustrate the height-balancing property of red-black and AVL trees.

Let $T$ be a BST histogram of a true color image. Let $d_T$ be the depth of a node in the BST. We define the structural query time of $T$:

$$T_Q = \sum_{n \in T} (d_T(n) + 1) \cdot c_T \cdot n.count\,, \tag{5}$$

where $c_T = 2$ is the number of comparisons made at each node. Hence, $T_Q$ is the average number comparisons required to search $T$ for all the pixels in $f$. Since $H_f(\boldsymbol{c})$ accurately represents the probability that $\boldsymbol{c}$ will be queried, we would like to construct our BSTs such that $T_Q$ is minimized. Clearly if all colors appeared equally often, then height-balancing would be the solution; otherwise, we may want to sacrifice height in order for nodes with high counts to be near the top of the tree. The nodes of the AVL tree histogram drawn in Color Plate F have been scaled by $\log_2(H_f)$. Many of the more popular colors of this AVL tree are near the bottom of the tree, thus increasing its structural query time. Using dynamic programming, one can statically build such an *optimal* BST in $O(N_p^3)$ [11]. The drawback to the dynamic programming algorithm is that the colors and their counts need to be known a priori.

Ideally, we would like to dynamically build optimal BSTs. This leads us to treaps [8]. Our truecolor histogram implementation of treaps simultaneously maintains the BST property on `key`(*node.c*) **and** the max heap property on *node.count*. Thus, the most popular color in the histogram will always be stored at the root. The empirical results given in Table 3 show that red-black trees and AVL trees build the most efficient BSTs and that the red-black trees are the most efficient to build. However, for small $RGB_5$, the distribution of the counts on the colors is such that treaps become competitive with red-black trees. Color Plate G illustrates how treaps are built such that colors with large counts are higher in the tree.

| | | **$RGB_8$** | | | **$RGB_7$** | | | **$RGB_6$** | | | **$RGB_5$** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $h$ | $t_B$ | $t_Q$ | $h$ | $t_B$ | $t_Q$ | $h$ | $t_B$ | $t_Q$ | $h$ | $t_B$ | $t_Q$ |
| BST | *Windsails* | 96 | 37.67 | 65.45 | 83 | 35.45 | 59.81 | 65 | 29.32 | 49.13 | 45 | 22.83 | 38.31 |
| | Max | 703 | 55.13 | 108.44 | 529 | 50.98 | 96.77 | 347 | 41.23 | 77.71 | 184 | 29.59 | 55.56 |
| | Min | 38 | 5.12 | 9.55 | 38 | 4.53 | 7.96 | 38 | 3.15 | 5.35 | 36 | 2.19 | 3.33 |
| | Avg | 131 | 30.61 | 54.72 | 113 | 29.27 | 52.19 | 82 | 23.80 | 42.11 | 54 | 17.68 | 29.97 |
| Red-black | *Windsails* | 19 | 20.86 | 28.51 | 19 | 20.27 | 28.29 | 18 | 17.76 | 25.81 | 15 | 15.25 | 22.32 |
| | Max | 21 | 22.16 | 29.31 | 21 | 21.17 | 28.95 | 19 | 18.38 | 25.94 | 16 | 16.04 | 22.60 |
| | Min | 11 | 2.45 | 3.31 | 12 | 2.44 | 3.65 | 11 | 2.11 | 2.99 | 10 | 1.94 | 2.70 |
| | Avg | 18 | 15.77 | 22.23 | 18 | 14.96 | 21.46 | 16 | 13.08 | 19.25 | 13 | 11.29 | 16.42 |
| AVL | *Windsails* | 19 | 32.08 | 28.56 | 18 | 31.18 | 28.56 | 17 | 25.94 | 25.63 | 14 | 22.25 | 22.14 |
| | Max | 21 | 34.38 | 29.49 | 20 | 32.27 | 29.14 | 17 | 27.21 | 26.18 | 15 | 22.75 | 22.14 |
| | Min | 11 | 3.75 | 3.76 | 10 | 3.43 | 3.17 | 10 | 3.15 | 3.03 | 10 | 2.81 | 2.52 |
| | Avg | 17 | 23.96 | 22.10 | 16 | 22.38 | 21.56 | 15 | 19.22 | 19.15 | 12 | 16.79 | 16.58 |
| Treap | *Windsails* | 72 | 27.7 | 40.59 | 67 | 26.77 | 39.63 | 56 | 21.22 | 30.31 | 42 | 16.45 | 23.57 |
| | Max | 703 | 31.87 | 54.19 | 192 | 29.36 | 44.74 | 102 | 25.76 | 41.64 | 66 | 17.72 | 27.30 |
| | Min | 29 | 2.10 | 2.64 | 30 | 2.05 | 2.55 | 29 | 1.88 | 2.32 | 18 | 1.75 | 2.08 |
| | Avg | 93 | 21.21 | 31.56 | 65 | 18.84 | 27.04 | 53 | 15.57 | 22.15 | 39 | 12.13 | 16.49 |

Table 3. BST statistics for the image test set.

## *4.4 Spatial Subdivision Methods*

The *spatial subdivision* methods discussed in this section are based on a decomposition of $RGB_p$ into smaller pieces called *partitions* or *cells*. Many strategies exist for determining the structure of spatial subdivisions. 2D arrays are a uniform subdivision of $RGB_p$ into $2^p \times 2^p$ partitions, each of which contains at most $2^p$ colors.

Hierarchical subdivisions such as octrees and *k*-d trees are constructed by recursively partitioning $RGB_p$ into smaller and smaller subdomains. We begin with one cell (the root at depth 0) whose domain is $RGB_p$ and whose elements are stored in a single bucket. When the number of elements in a bucket exceeds the maximum bucket size ($B$), the cell is partitioned into smaller pieces, and the elements of the overflowing bucket are inserted into the unique partition whose domain includes the element. The space utilization of spatial subdivisions is defined as $\lambda = O/M$, where $M$ is the number of cells and $O$ is the number of non-empty cells.

### *4.4.1 2D Arrays*

Equation 5 is used to calculate the structural query time of 2D arrays by interpreting the depth of a node to be its position in the linked list (starting at position 0) and letting $c_T$=1. The empirical results given in Tables 2 and 4 show that on average ($5 \leq p \leq 8$), the space utilization of 2D arrays is 638% better than 3D arrays; however, this improved $\lambda$ incurs larger build and query times. Table 4 shows that doubly linked lists, red-black trees and treaps are fairly competitive data structures for 2D array buckets. For our data image set, red-black trees performed the best in terms of build time, and treaps were the best in terms of query time. Hence, if the histogram is built once, but queried many times, treaps may be the better choice for a bucket structure.

| | | **$RGB_8$** | | | **$RGB_7$** | | | **$RGB_5$** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $\lambda$ | $t_B$ | $t_Q$ | $\lambda$ | $t_B$ | $t_Q$ | $\lambda$ | $t_B$ | $t_Q$ |
| List | *Windsails* | 0.379 | 9.52 | 8.38 | 0.473 | 10.75 | 10.98 | 0.546 | 9.94 | 9.92 |
| | Max | 0.433 | 11.06 | 10.12 | 0.781 | 13.73 | 14.190 | 1.000 | 9.94 | 9.92 |
| | Min | 0.009 | 1.50 | 1.17 | 0.034 | 1.32 | 1.170 | 0.128 | 1.18 | 1.14 |
| | Avg | 0.228 | 7.85 | 7.20 | 0.341 | 8.61 | 8.749 | 0.448 | 7.10 | 7.05 |
| Red-black | *Windsails* | 0.379 | 9.76 | 8.47 | 0.473 | 10.06 | 9.68 | 0.546 | 8.05 | 8.98 |
| | Max | 0.433 | 10.65 | 9.43 | 0.781 | 10.78 | 10.810 | 1.000 | 8.58 | 9.62 |
| | Min | 0.009 | 1.42 | 1.15 | 0.034 | 1.32 | 1.170 | 0.128 | 1.16 | 1.17 |
| | Avg | 0.228 | 7.75 | 7.08 | 0.341 | 7.747 | 7.786 | 0.448 | 6.47 | 7.03 |
| Treap | *Windsails* | 0.379 | 11.57 | 8.17 | 0.473 | 12.05 | 9.14 | 0.546 | 9.82 | 8.09 |
| | Max | 0.433 | 12.75 | 9.22 | 0.781 | 13.27 | 10.420 | 1.000 | 10.13 | 8.61 |
| | Min | 0.009 | 1.66 | 1.16 | 0.034 | 1.57 | 1.150 | 0.128 | 1.45 | 1.18 |
| | Avg | 0.228 | 9.28 | 6.96 | 0.341 | 9.35 | 7.338 | 0.448 | 7.93 | 6.55 |

Table 4. 2D array statistics for the image test set.

### *4.4.2 Octrees*

Octrees are a hierarchical subdivision method which subdivides the cells (*octants*) using three cut-planes which are orthogonal to each of the R, G, B axes. The point at which the three cut-planes intersect is application dependent. The octrees discussed in this section use the octant's center; thus, the octant is subdivided into eight equal-sized subcubes.

Color Plates I and J illustrates the adaptive nature of octree partitioning. The number of cells contained in the octree depends on *B* and the distribution of the colors in 3D space. Sparse regions of $RGB_p$ are represented by few cells, while dense regions are represented by many. Additionally, the octree clusters neighboring colors in the RGB cube into a common octant. This clustering is a desirable property for color quantization and is the basis of the oct-cut algorithm proposed by Roytman and Gotsmann [15].

Let *S* be an octree histogram of a truecolor image. Let $d_S$ be the depth of a bucket in *S*. We define the structural query time of *S*:

$$T_Q = \sum_{b \in S} \sum_{n \in b} \left( (d_S(b)+1) \cdot c_S + (d_T(n)+1) \cdot c_T \right) \cdot n.count, \tag{6}$$

where $c_S = 4$. The empirical results given in table 4 and 5 show that octrees obtain better space utilization than 2D arrays; but once again, this improved λ comes at the cost of larger build and query times. Table 4 shows that red-black trees and treaps are fairly competitive data structures for octree buckets.

| | | **$RGB_8$** | | | **$RGB_7$** | | | **$RGB_5$** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $\lambda$ | $t_B$ | $t_Q$ | $\lambda$ | $t_B$ | $t_Q$ | $\lambda$ | $t_B$ | $t_Q$ |
| List | *Windsails* | 0.797 | 45.09 | 23.9 | 0.723 | 44.27 | 23.34 | 0.721 | 36.31 | 20.73 |
| | Max | 0.833 | 46.40 | 25.15 | 0.780 | 48.96 | 29.46 | 0.812 | 41.04 | 25.68 |
| | Min | 0.416 | 4.57 | 2.31 | 0.360 | 5.49 | 3.44 | 0.421 | 3.82 | 2.08 |
| | Avg | 0.685 | 35.36 | 18.37 | 0.645 | 34.18 | 19.10 | 0.669 | 26.24 | 16.14 |
| Red-black | *Windsails* | 0.797 | 27.17 | 15.77 | 0.785 | 26.11 | 16.01 | 0.709 | 17.23 | 13.93 |
| | Max | 0.833 | 29.29 | 16.45 | 0.828 | 27.49 | 16.40 | 0.832 | 18.01 | 14.71 |
| | Min | 0.416 | 2.80 | 1.930 | 0.416 | 2.47 | 1.96 | 0.488 | 1.99 | 1.80 |
| | Avg | 0.685 | 20.84 | 12.84 | 0.679 | 19.35 | 12.75 | 0.691 | 13.28 | 11.32 |
| Treap | *Windsails* | 0.797 | 28.37 | 15.11 | 0.785 | 27.05 | 14.89 | 0.709 | 17.27 | 12.22 |
| | Max | 0.833 | 31.20 | 16.10 | 0.828 | 29.03 | 15.74 | 0.832 | 18.39 | 13.77 |
| | Min | 0.416 | 2.81 | 1.90 | 0.416 | 2.44 | 1.82 | 0.488 | 2.00 | 1.70 |
| | Avg | 0.685 | 21.74 | 12.15 | 0.679 | 20.08 | 11.91 | 0.691 | 13.33 | 9.99 |

Table 5. Octree statistics for the image test set, *B*=64.

### *4.4.3 k-d Trees*

The hierarchical method used to subdivide *k*-d tree cells is much more flexible than the octree method. Instead of subdividing the cells with three cut-planes, only one orthogonal cut-plane is selected. Various methods exist for determining which axis to cut and where to position the cut-plane. The *k*-d trees discussed in this section were constructed by cutting the axis with the largest range at its center. Balasubramanian and Allebach [1] use *k*-d trees to perform fast pairwise nearest neighbor searches. Their *k*-d trees use linked lists as the bucket structure and are built by inserting (color, count) pairs from a previously built 2D array histogram. Linked lists were a reasonable choice, because they prequantized the image data and fixed the maximum bucket size at 8. We propose using larger maximum bucket sizes, and hence using red-black trees and treaps as the bucket structure. In addition, we build the *k*-d tree histograms directly from the truecolor images.

Color Plates I-L illustrate how much more adaptive *k*-d trees are than octrees. The shapes of the *k*-d tree cells vary and adapt to the statistics of the histogram, thus enabling the cells of the *k*-d tree to better cluster neighboring colors in the RGB cube.

By letting $c_S$=2, Equation 6 is used to calculate the structural query time of a *k*-d tree. All of the *k*-d trees summarized in Table 6 had optimal space utilization. However, tables 5 and 6 show that *k*-d trees require more time to build and query than octrees. Table 6 shows that red- black trees and treaps are competitive data structures for *k*-d tree buckets, but that red-black trees are better, particularly for $p \geq 7$.

| | | **RGB$_8$** | | | **RGB$_7$** | | | **RGB$_5$** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $M$ | $t_B$ | $t_Q$ | $M$ | $t_B$ | $t_Q$ | $M$ | $t_B$ | $t_Q$ |
| Red-black | Windsails | 544 | 30.49 | 19.58 | 461 | 28.85 | 19.40 | 34 | 18.74 | 16.73 |
| | Max | 1366 | 38.01 | 20.89 | 832 | 29.98 | 20.17 | 50 | 18.88 | 16.73 |
| | Min | 6 | 3.450 | 2.45 | 6 | 3.09 | 2.43 | 3 | 2.51 | 2.10 |
| | Avg | 334 | 22.39 | 15.44 | 232 | 20.27 | 15.27 | 17 | 13.95 | 13.09 |
| Treap | Windsails | 544 | 34.95 | 20.83 | 461 | 32.75 | 20.16 | 34 | 18.9 | 15.36 |
| | Max | 1366 | 45.85 | 24.27 | 832 | 35.11 | 22.31 | 50 | 21.44 | 17.03 |
| | Min | 6 | 3.31 | 2.30 | 6 | 2.89 | 2.32 | 3 | 2.33 | 2.07 |
| | Avg | 334 | 25.65 | 16.47 | 232 | 22.87 | 15.46 | 17 | 14.38 | 12.17 |

Table 6. *k*-d tree statistics for the image test set, *B*=256.

## 5. Summary

We have made a visual exploration of the space and time requirements of 3D data structures used to store truecolor image histograms. We have also seen the clustering property of spatial subdivisions. We proposed using treaps and red-black trees instead of doubly linked lists and BSTs for spatial subdivision bucket structures. We discovered that for our image test set, red-black trees and treaps proved to be competitive bucket structures for histograms built using spatial subdivision techniques, but that red-black trees were the best overall when $p \geq 7$ is desired.

Next we hope to visually discover better heuristics for color quantization by incorporating our visualizations of truecolor image histograms into animations of color quantization techniques. We plan to make our animations available to other researchers via the World Wide Web.

Since presenting this paper at COMPUGRAPHICS ’97 [13] we have made several key implementation changes which have resulted in a significant reduction in the empirical timings: 1) the histogram class hierarchy was simplified, 2) the number of functions declared as virtual was reduced, and 3) the RGB class was changed to contain an array of three `unsigned chars` vice three `unsigned chars`. For instance, the time now required to build a BST for *Windsails* in $RGB_8$ has been reduced from 37.67 to 12.49 seconds. These improvements will be reported in a future paper.

## Acknowledgements

The author is grateful for the generous support provided by the United States Naval Academy, without which this research would not be possible. The author also thanks Dr. William D. Withers for introducing her to the interesting field of color quantization. The author is extremely appreciative of the thoughtful encouragement Dr. Andrew T. Phillips and Dr. Harold P. Santo provided her.

## References

[1] Raja Balasubramanian and Jan P. Allebach, A New Approach to Palette Selection for Color Images, *The Journal of Imaging Science and Technology*. **17(6)** (December 1991) 284-90.

[2] Shahid H. Bokhari, Thomas W. Crockett, and David M. Nichol, Binary Dissection: Variants & Applications, **ICASE 97-29**, Instititue for Computer Applications in Science and Engineering, Hampton, Virginia, 1997.

[3] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis, Algorithms for Drawing Graphs: an Annotated Bibliography, *Computational Geometry: Theory & Applications*. **4** (1994) 235-82.

[4] Chen Ding and Prabhaker Mateti, A Framework for the Automated Drawing of Data Structure Diagrams, *IEEE Transactions on Visualization and Computer Graphics*. **16(5)** (May 1990) 543-57.

[5] Ashim Garg and Roberto Tamassia, GIOTTO3D: A System for Visualizing Hierarchical Structures in 3D, in *Symposium on Graph Drawing '96 Proceedings, LNCS-1190*, (Berkeley, California, 1996) 193-200.

[6] Paul S. Heckbert, Color Image Quantization for Frame Buffer Display, *Computer Graphics*. **16(3)** (July 1982) 297-304.

[7] Kurt Mehlhorn and Stefan Näher, LEDA: A Platform for Combinatorial and Geometric Computing, *Communications of the ACM*. **38(1)** (January 1995) 96-102.

[8] Rajeev Motwani and Prabhakar Raghavan, *Randomized Algorithms*, (Cambridge University Press, New York, 1995).

[9] James D. Murray and William vanRyper, *Encyclopedia of Graphics File Formats*, (O'Reilly & Associates, Inc., Sebastopol, CA, 1994).

[10] Marc A. Najork and Marc H. Brown, Obliq-3D: A High-Level, Fast Turnaround 3D Animation System, *IEEE Transactions on Visualization and Computer Graphics*. **1(2)** (June 1995) 175-93.

[11] Richard E. Neapolitan and Kumarss Naimipour, *Foundations of Algorithms*, (D.C. Heath and Company, Lexington, Massachusetts, 1996).

[12] Soo-Chang Pei and You-Shen Lo, Color image compression and limited display using self-organization Kohonen map, *IEEE Transactions on Circuits and Systems for Video Technology*. **8(2)** (April 1998) 191-205.

[13] Paula J. Reitan, Visualization of Truecolor Image Histograms, in *COMPUGRAPHICS ’97*, (GRASP, Vilamoura, Portugal 1997) 320-9.

[14] Edward M. Reingold and John S. Tilford, Tidier Drawings of Trees, *IEEE Transactions on Software Engineering*. **7(2)** (March 1981) 223-8.

[15] Evgeny Roytman and Craig Gotsman, Dynamic Color Quantization of Video Sequences, *IEEE Transactions on Visualization and Computer Graphics*. **1(3)** (September 1995) 274-86.

[16] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card, Cone Trees: Animated 3D Visualization of Hierarchical Information, in *Proceedings of CHI '91*, (New Orleans, Louisiana, 1991) 189-94.

[17] Windsails on the Sand - Negril, Jamaica, in *Carribean* Professional Photo CD-ROM, (Corel, Ottawa, Canada, 1994).

[18] Jefferey A. Shufelt, Color Image Quantization Enhancement Techniques, **CMU95-100**, Computer Science Department, Carnegie Mellon University, 1995.

[19] Tolga Tasdizen, Lale Akarun, and Cem Ersoy, Color Quantization with Genetic Algorithms, *Signal Processing: Image Communication*. **12** (1998) 49-57.

[20] Ching-Yung Yang and Ja-Chen Lin, RWM-Cut for Color Image Quantization, *Computers and Graphics*. **20(4)** (1996) 577-88.

## Vitae

**Paula J. Reitan** is a Ph.D. candidate in the Computer Science and Electrical Engineering Department at the University of Maryland, Baltimore County. Her dissertation research is in the development of a heterogeneous 3D data structure for color image quantization. In 1986 she received the B.S. degree in computer science from the United States Naval Academy where she teaches part-time. She received the M.S. degree in computer science from the Johns Hopkins University in 1987. She is a member of the ACM.